Artificial and Computational Intelligence

AIMLCZG557

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

ACI Team

innovate achieve lead

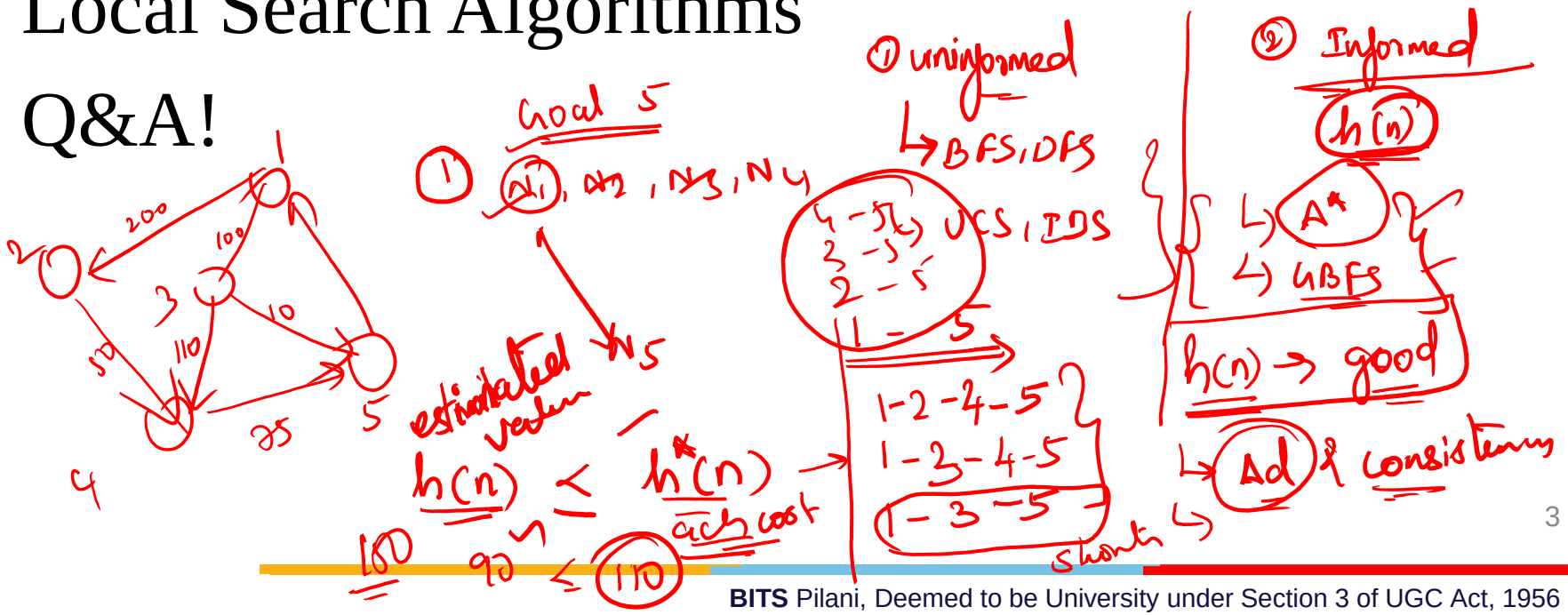
BITS Pilani

Pilani|Dubai|Goa|Hyderabad

AIMLCZG557 – CS#5
Designing Heuristics & Introduction to Local Search Algorithms

Agenda for CS #5

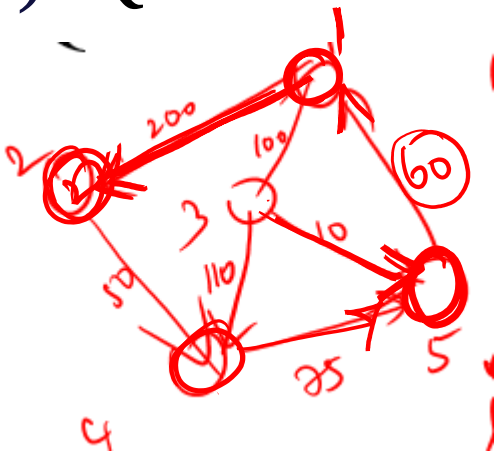
1) Designing Heuristics
2) Heuristics from Relaxed problems, Pattern DB..
3) Use case: Reference Paper
4) Local Search Algorithms
5) Q&A!



BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Agenda for CS #5

1) Designing Heuristics
2) Heuristics from Relaxed problems, Pattern DB..
3) Use case: Reference Paper
4) Local Search Algorithms
5) Q&A!



① → Incoming edges 5 → 1

h(1) ⇒ 65
h(5) ⇒ 90

90 - 65 ≤ 60

h(5) - h(1) ≤ g(5,1)

⇓

h(5) ≤ g(5,1) + h(1)

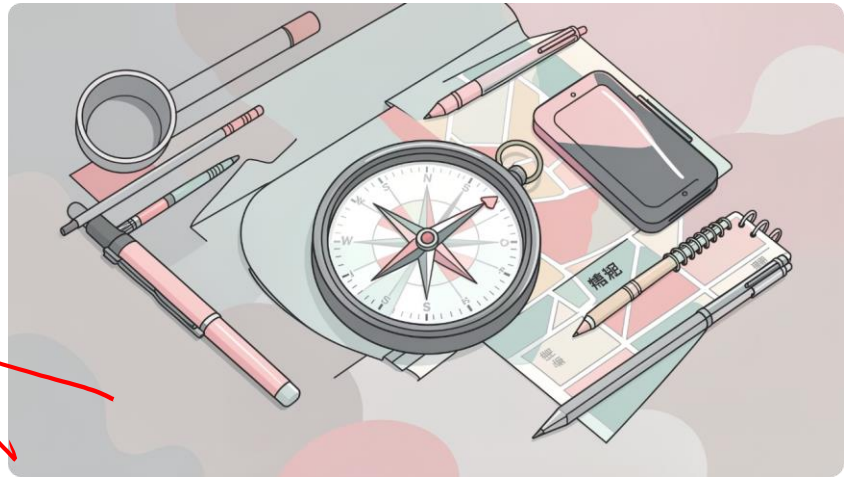
35 ≤ 60 ✓

{3-5
{4-5

Introducing Heuristics: The "Good Enough" Guiding Star

What is a Heuristic?

A practical rule of thumb or educated guess that estimates the cost from current state to the goal. Derived from Greek "heuriskein" – "to find" or "to discover."



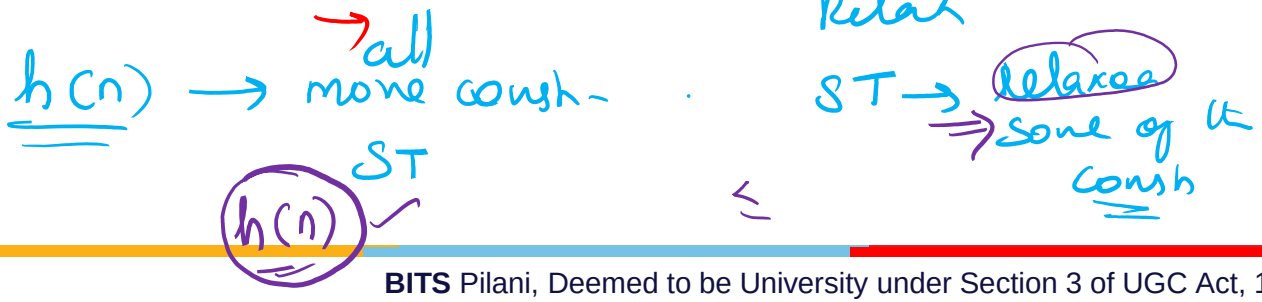
Key Insight

Prioritize exploration of paths that **seem most promising** rather than exploring all possibilities blindly.

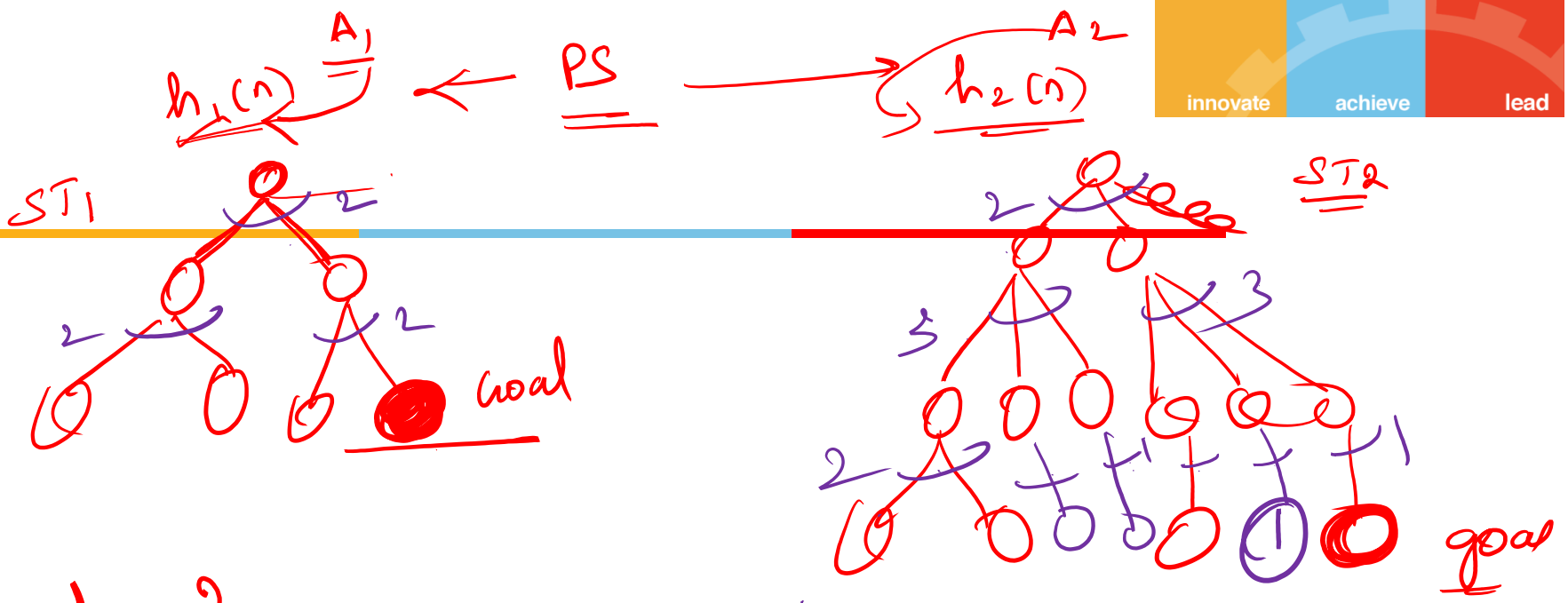
Like using a compass and map in a maze instead of wandering aimlessly

Heuristics Design

- As heuristic functions determine the search strategies and thereby their complexities,
  - it is important that we identify better heuristic functions
  - While maintaining Admissible and Consistency constraints
- A good heuristic design can be selected using:
  - Effective Branching Factor
  - Generating admissible heuristics from relaxed problems
  - Generating admissible heuristics from sub problems
  - Learning Heuristics from Experience



BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956



d = 2

bF < min / max [2, 2, 2] = 2 / Avg [2,]

b13 = 2/1

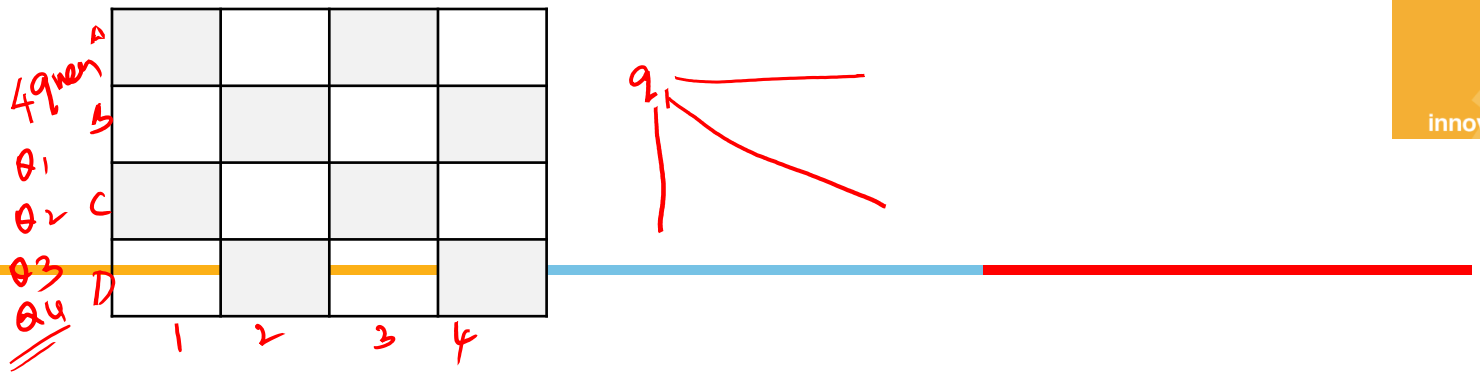
d = 3

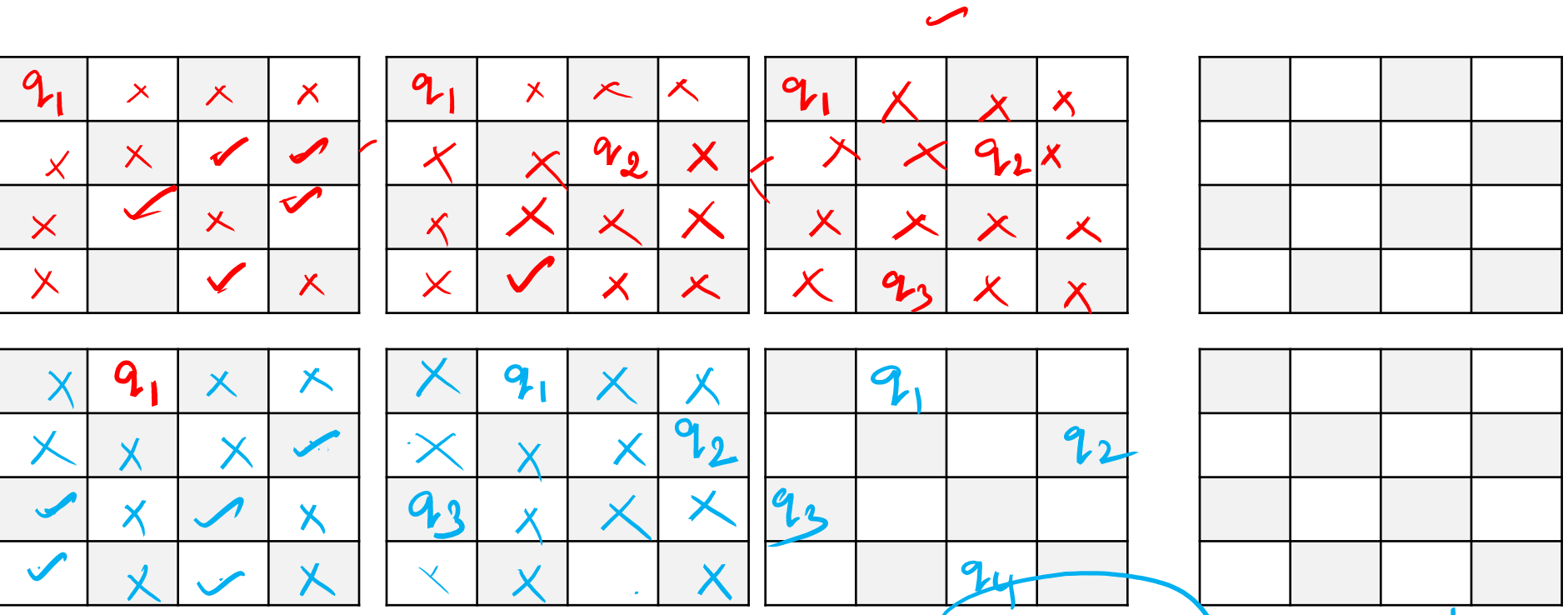
bF - max 3 [2, 3, 3, 2, (- .)

Avg : 5/9 =

=

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956





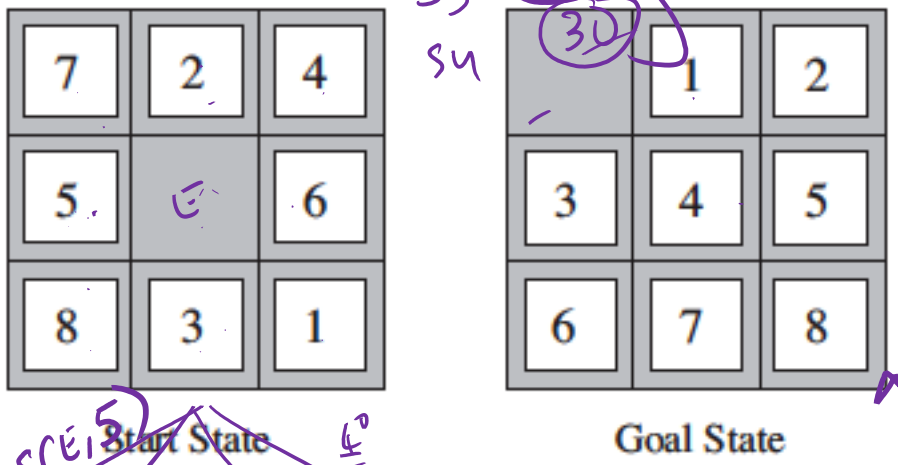
real PS

3 constraints:
1. not in same row
2. " " " column
3. " " " diagonal

3 constraints relax → relaxed Problem → 2 constraints

Activity: Heuristics

Provide atleast 2 heuristics for this problem!



h_1 (n) = number of misplaced tiles, admissible

h_2 (n) = sum of Manhattan distances of tiles from their respective goal positions, admissible

Test:

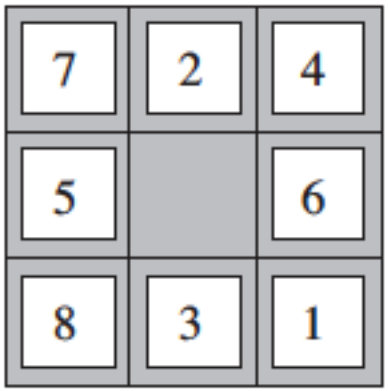
h_1 (S) = ? 8

h_2 (n) = ? 3+1+2+2+2+3+3+2 = 18

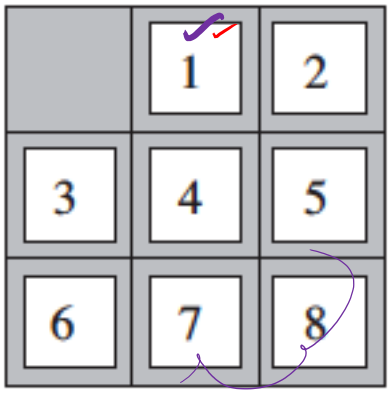
BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Activity: Heuristics

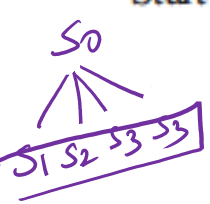
Provide atleast 2 heuristics for this problem!

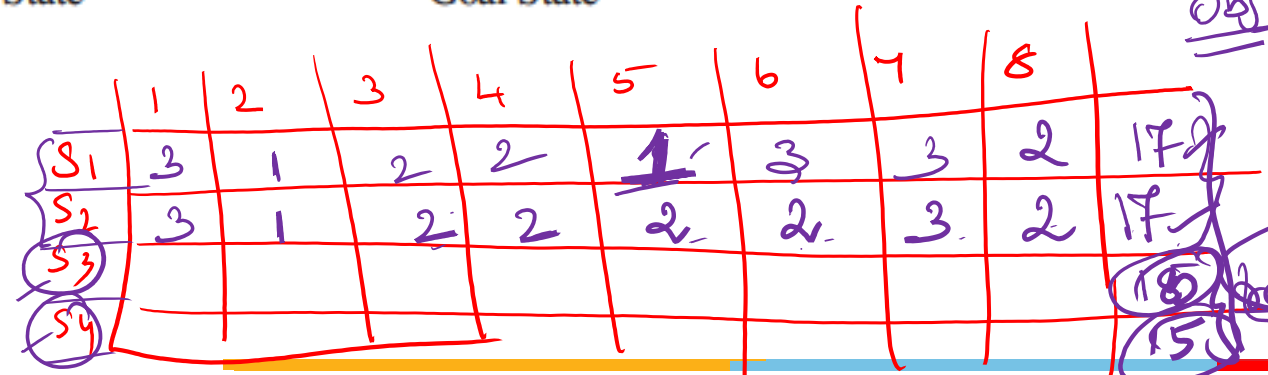


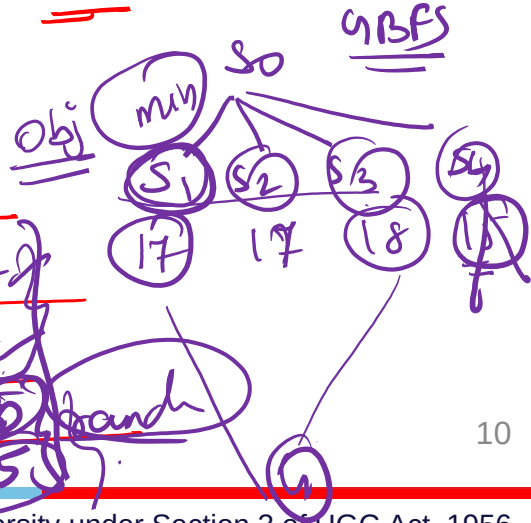
Start State



Goal State

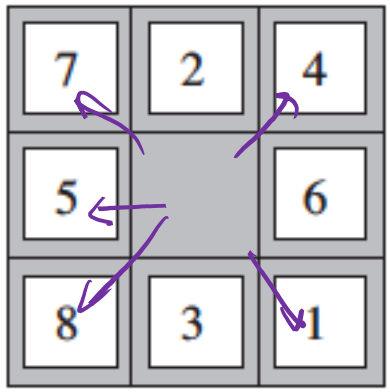




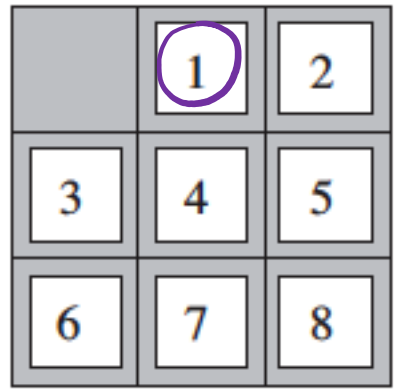


BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

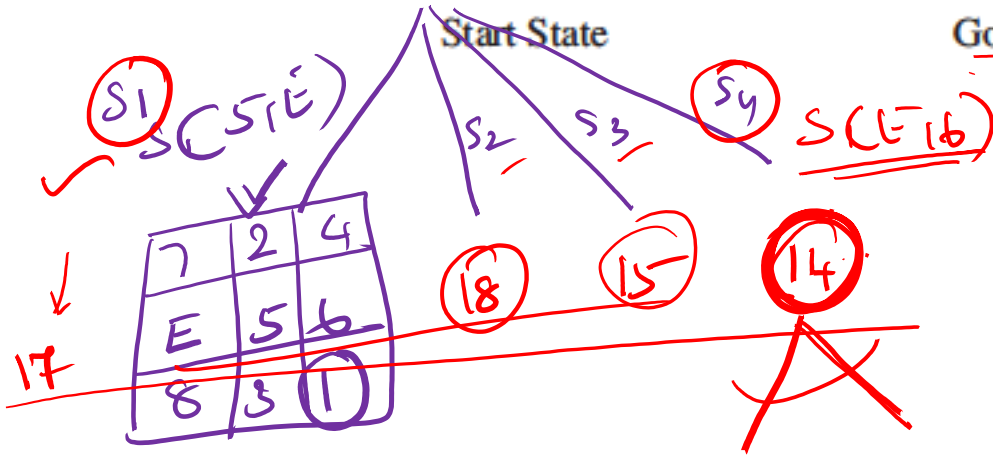
Activity: Heuristics



Start State

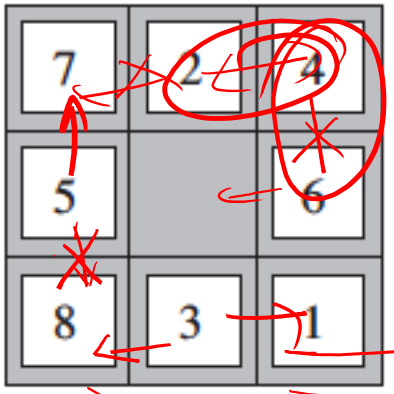


Goal State

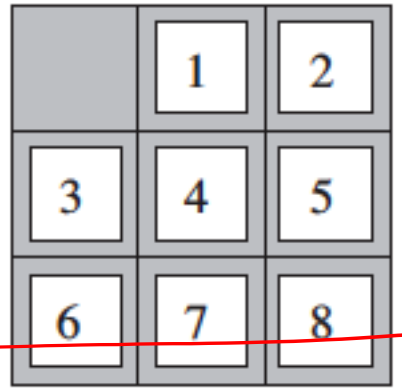


BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

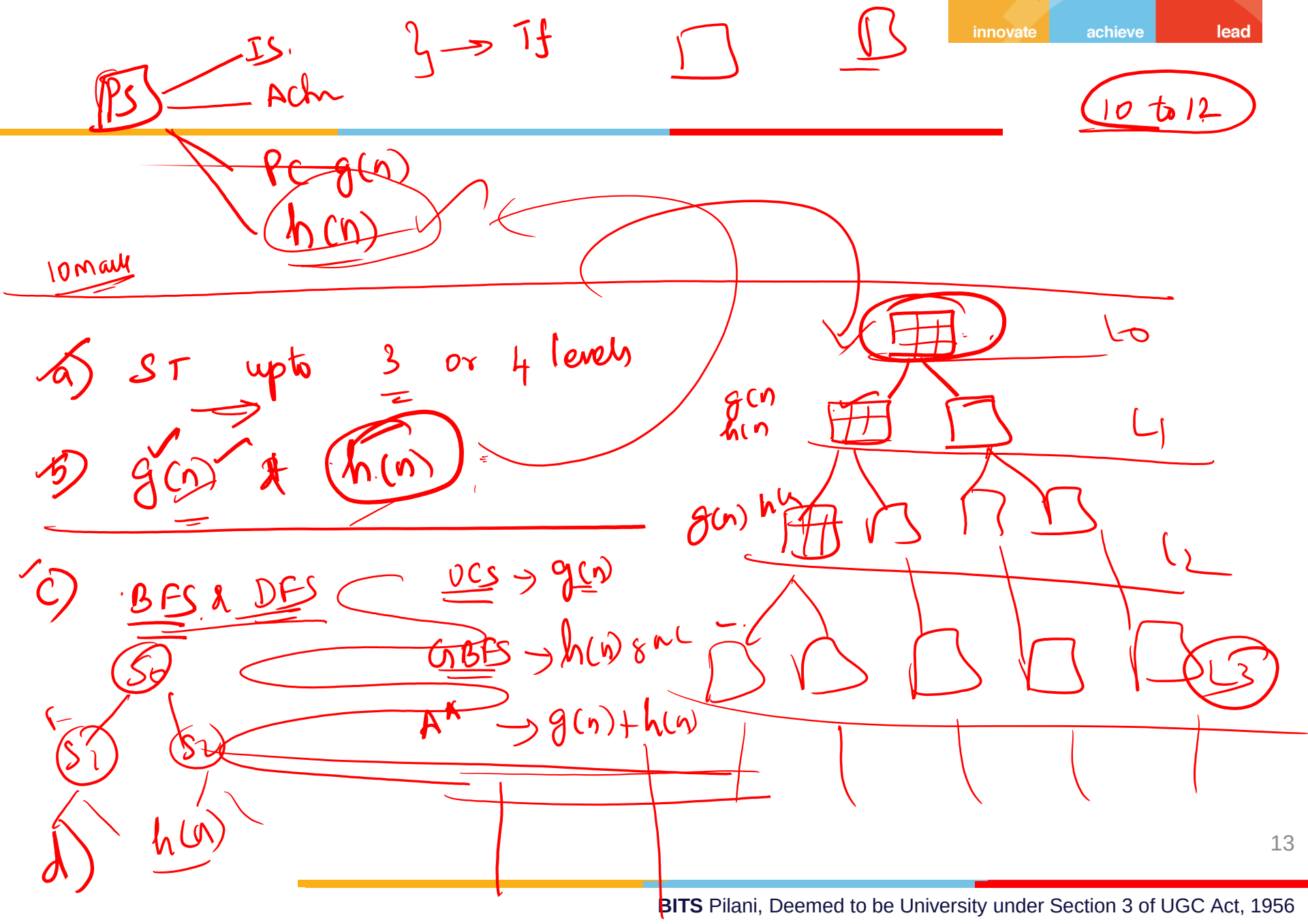
Activity: Heuristics



Start State



Goal State


IS.
Achn
PS
10 to 12
PC g(n)
h(n)
10 mark
a) ST upto 3 or 4 levels
b) g(n) & h(n)
c) BFS & DFS
UCS → g(n)
GBFS → h(n) & nc
A* → g(n)+h(n)
S0
S1
S2
h(n)
d)
L0
L1
L2
L3
g(n)
h(n)

Need of Heuristics

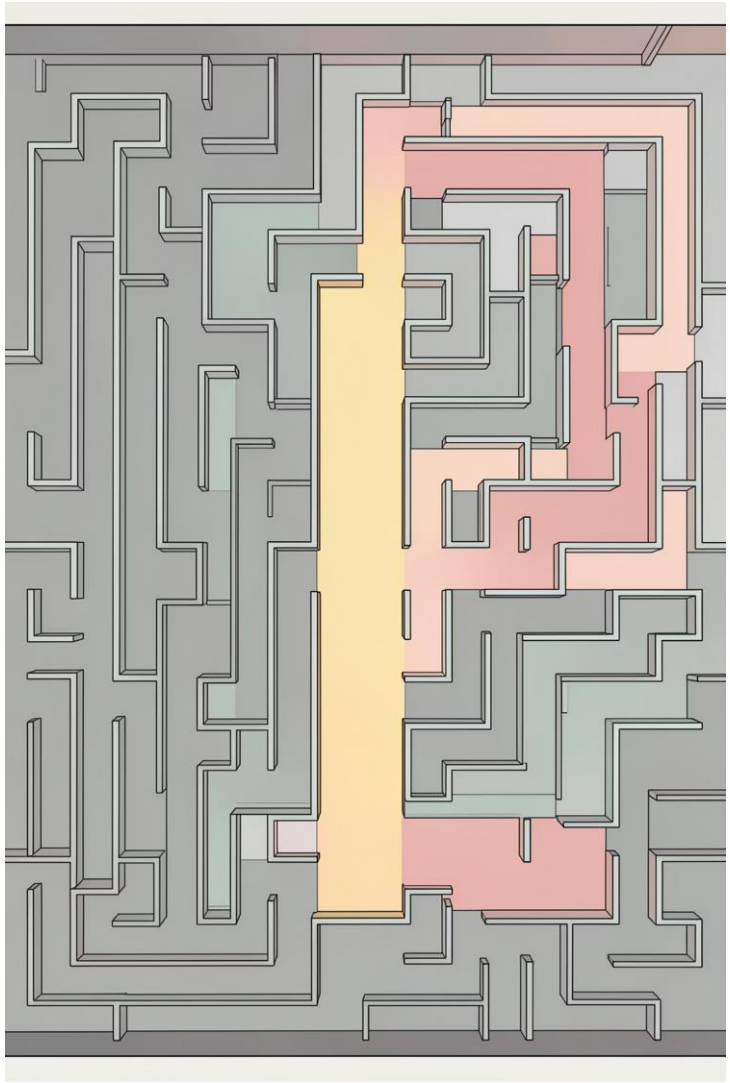
- Humans rarely search blindly.
- We use educated guesses.
- AI systems also need estimates to guide search.
- These estimates are called **heuristics**.

Example: While driving, we choose roads that appear to move us closer to the destination.

Real-Life Examples:

- Google Maps suggests shortest routes
- Chess engines evaluate board positions
- Doctors infer diseases from symptoms
- Robots estimate distance to targets

The Power of Informed Search: A* and Beyond



01

Best-First Search

Expands nodes based on heuristic evaluation function

02

Cost Function

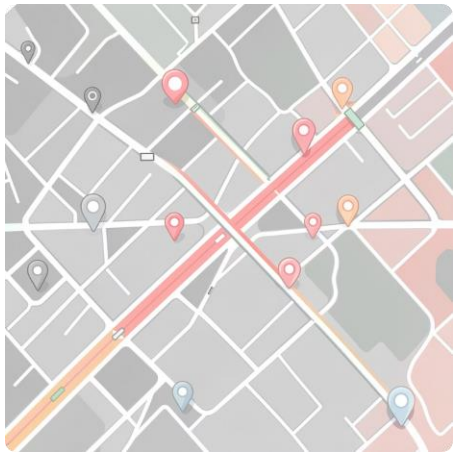
f(n) = g(n) + h(n) where g(n) is actual cost from start and h(n) is estimated cost to goal

03

A* Algorithm

Guarantees optimal path if heuristic is admissible (never overestimates true cost)

Real-World AI Applications: Where Heuristics Shine



Pathfinding

GPS navigation systems use Manhattan or Euclidean distance heuristics for route optimization

Game AI

Chess engines like Deep Blue evaluate board positions using sophisticated heuristic functions

Robotics

Robot navigation and motion planning rely on heuristics to find efficient collision-free paths

Logistics

Optimizing delivery routes and resource allocation in complex scheduling problems

Advanced Heuristic Techniques: Pushing the Boundaries

innovate achieve lead

IDA* (Iterative Deepening A*)

Memory-bounded version of A* for large state spaces, combining depth-first search with cost limits

SMA* (Simplified Memory-bounded A*)

Further memory optimization by dropping nodes when memory is full, retaining best paths

Combining Heuristics

Merging multiple heuristic functions through maximum operation for improved accuracy

Learning Heuristics

Deriving heuristics from relaxed problems, subproblems, pattern databases, or past experience

Mastering Heuristic Design

Foundation

Understanding heuristic properties: admissibility, consistency, and dominance relationships

Design Skills

Creating effective heuristic functions for novel AI problems and domains

Implementation

Building and analyzing search algorithms like A*, IDA*, and SMA* variants

Your Role: *Become architects of intelligent search, building more efficient and capable AI systems that can tackle real-world complexity.*

Heuristic Design

- Effective Branching Factor ($b*$)
  - If the algorithm generates N number of nodes, and the solution is found at depth d, then

$$N + 1 = 1 + (b^*) + (b^*)^2 + (b^*)^3 + \ldots + (b^*)^d$$

- Good Heuristics
  - Accurate, Fast to compute, Admissible, Consistent, Informative
- Notion of Relaxed Problems
  - A relaxed problem removes some constraints from the original problem, making it easier to solve. The cost of the relaxed problem becomes a lower bound on the original problem cost.
- Generating Admissible Heuristics

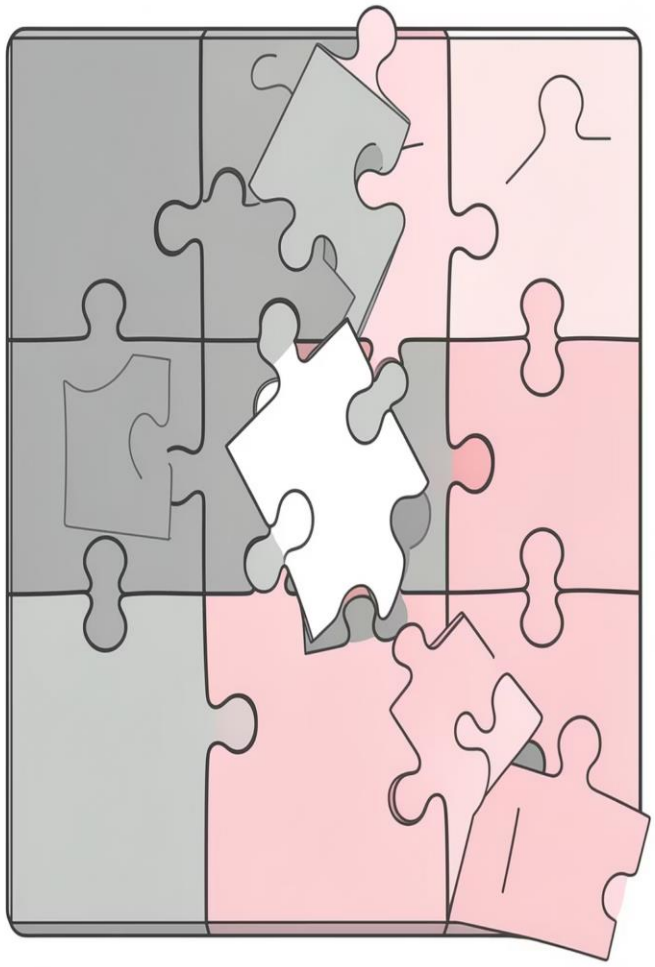
How do we design these heuristics?

- Relaxed Problems: We create a simpler version of the game by removing rules.
  - For example, if tiles could teleport or move through each other, the puzzle would be easy to solve. The cost of solving this "easy" version becomes our heuristic for the "hard" version.
- Pattern Databases: We solve a small part of the puzzle first (like just the first four tiles) and save every possible configuration and its cost in a database. When the AI is searching, it just looks up the answer in the table.
- Landmarks: Imagine you are using a GPS. It's too hard to calculate the path between every single street corner in the world. Instead, the system picks "landmark" cities and pre-calculates the distances to them.
  - By comparing where you are to a landmark, the AI can estimate the distance to your destination.

Generating Heuristics from Relaxed Problems

Idea: Solve a simplified version of the original problem where some constraints are removed. The cost to solve the relaxed problem becomes an admissible heuristic for the original problem.

1. **Original Problem**
   Move tiles to form specific configuration with movement constraints
2. **Relaxed Problem 1**
   Ignore tile movement constraints—tiles can move anywhere.
   Heuristic: Number of misplaced tiles
3. **Relaxed Problem 2**
   Allow tiles to move through each other. Heuristic:
   Manhattan distance (sum of distances from goal position)

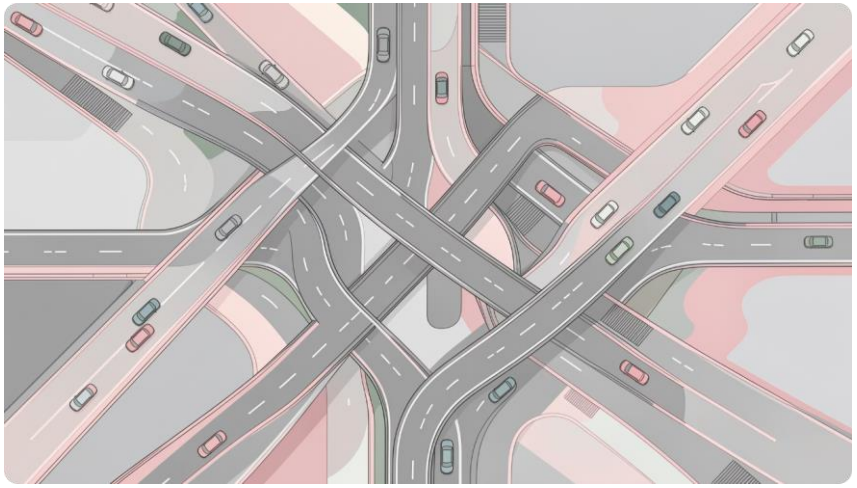


Key Benefit: Provides a lower bound on true cost, ensuring optimality in A* search while dramatically reducing explored nodes.

Generating Heuristics with Landmarks

Idea

Identify a set of “landmark” states that are easy to reach from many other states. The heuristic h(n) is based on the distance to the nearest landmark, providing informed estimates for large, complex graphs.



Example: Road Networks

Landmarks could be major cities or highway interchanges. The heuristic for a current location n is the shortest known distance from *n* to any landmark.

Advantage

Can provide more informed estimates than simple relaxed problems, especially in large graphs where traditional heuristics struggle.

Learning from Experience

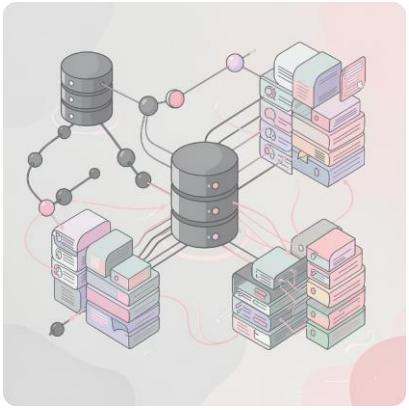
Can an AI learn to be a better searcher on its own? Yes!

- Metalevel Learning: This is when the AI “thinks about its own thinking”. It looks at the search paths it took and learns to avoid "dead ends" or unhelpful steps in the future.
- Feature Learning: An AI can solve thousands of random puzzles and look for patterns. It might notice that certain features, like how many tiles are out of order, are great predictors of the final cost and combine them into a single mathematical formula to guide its search.

.

Learning Heuristics from Experience

Instead of manually designing heuristics, machine learning discovers patterns from data, learning to predict costs from solved problem instances.

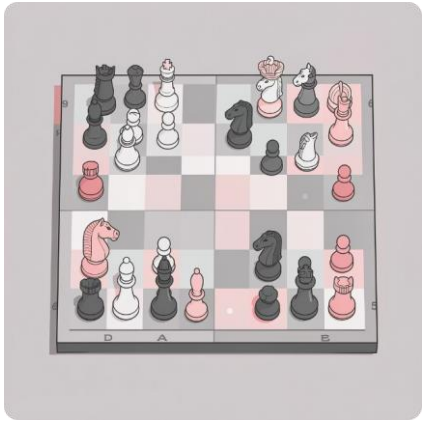


Pattern Databases

Pre-compute optimal solutions for smaller subproblems and use them to estimate costs for larger problems efficiently.

Neural Networks

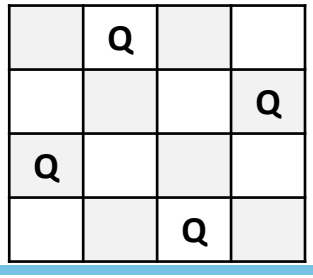
Train deep neural networks to predict the cost to the goal based on the current state representation.



Game Applications

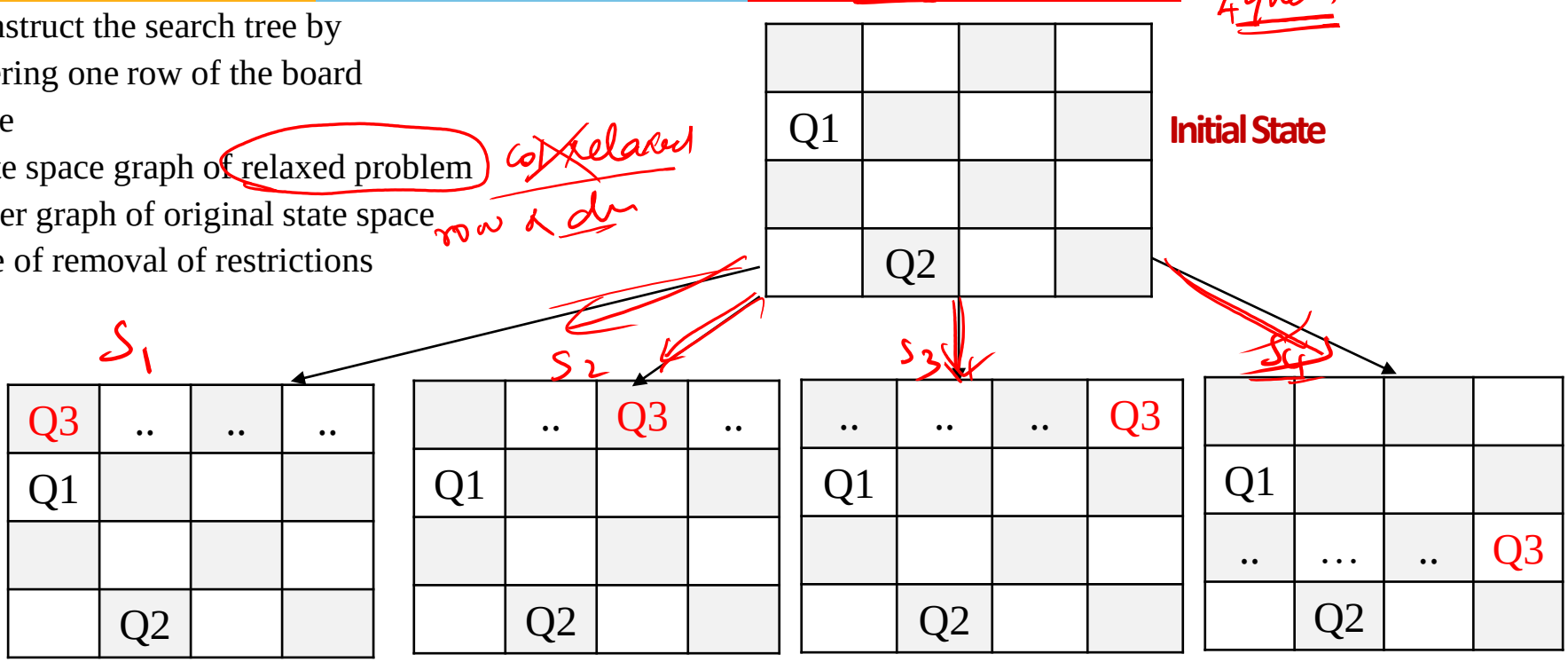
Learning heuristics for complex games like Go or Chess, where manual design is nearly impossible.

N-Queen



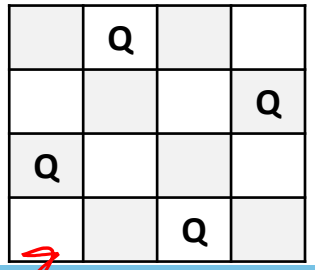
Goal State

- Construct the search tree by considering one row of the board at a time
- State space graph of relaxed problem is a super graph of original state space because of removal of restrictions

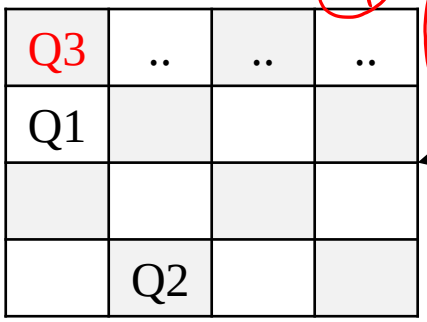


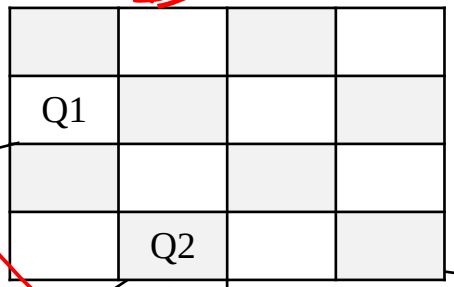
Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
< Xi , Yi >	Place in any non-occupied row in board		isValid Non-Attacking	Transition + Valid Queens	n!

N-Queen

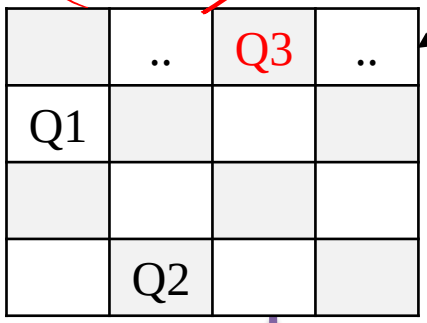


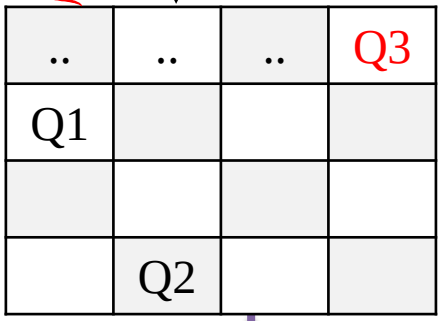
Goal State

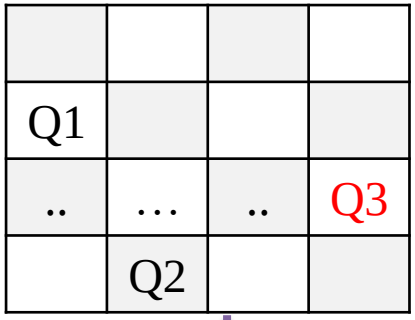




Initial State



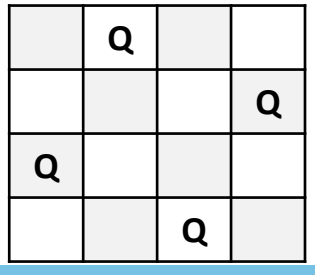




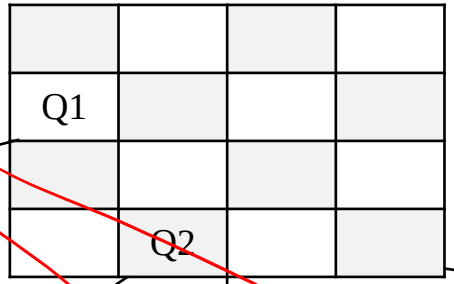
Possible Heuristic Design:

H1(n) : Number of Non conflicting Pairs of Queen (MAX)				
H2(n) : Number of Conflicting Pairs of Queen (MIN)				
H3(n) : Number of safest Queen (MAX)				

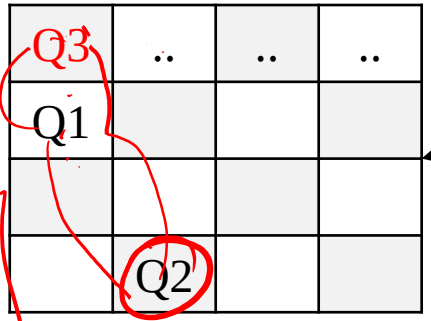
N-Queen

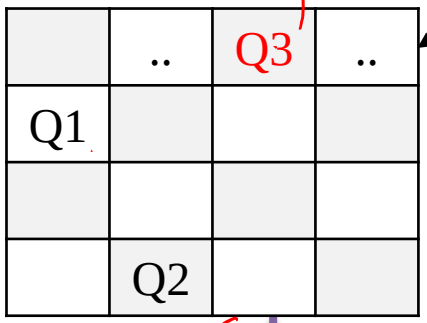


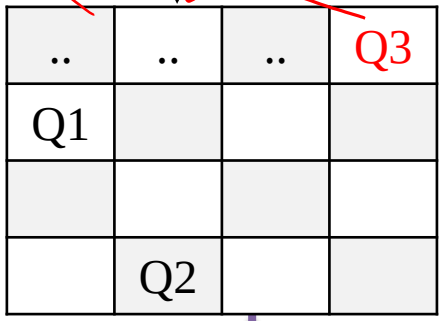
Goal State

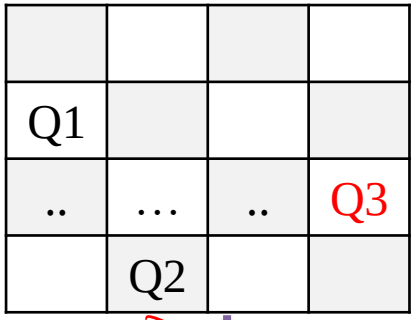


Initial State





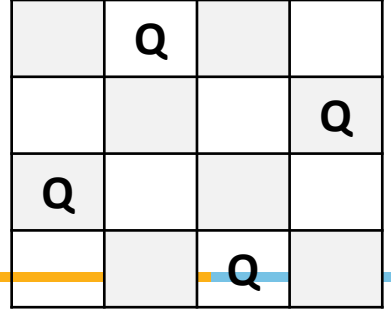




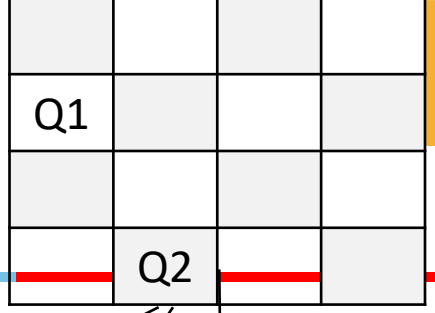
Possible Heuristic Design:

H1(n) : Number of Non conflicting Pairs of Queen (MAX)	2	3	3	3
H2(n) : Number of Conflicting Pairs of Queen (MIN)	1	0	0	0
H3(n) : Number of safest Queen (MAX)	1	3	3	3

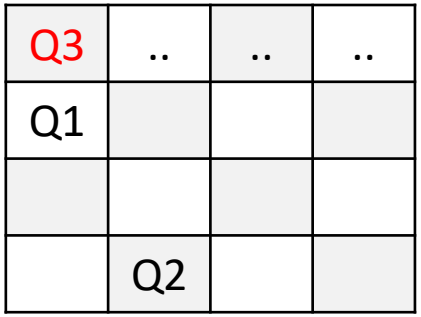
N-Queen

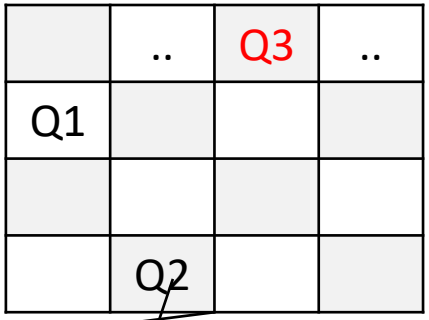


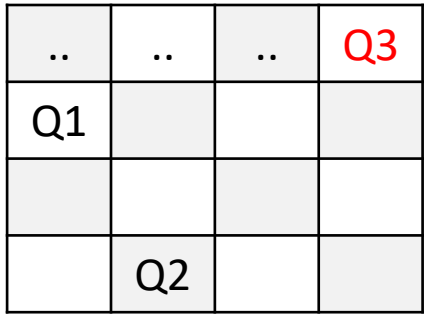
Goal State

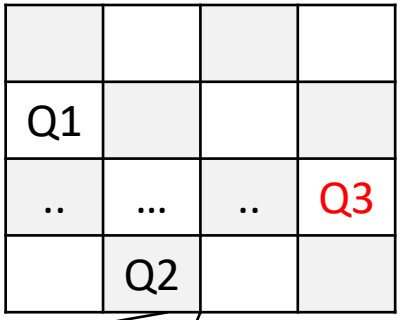


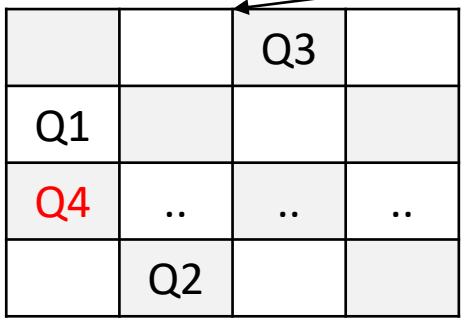
Initial State

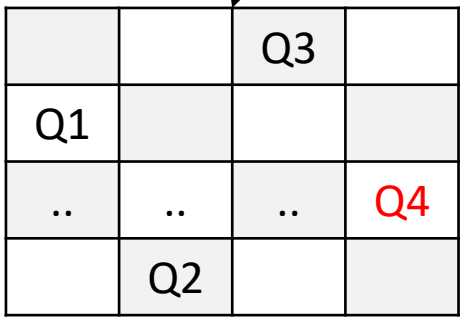


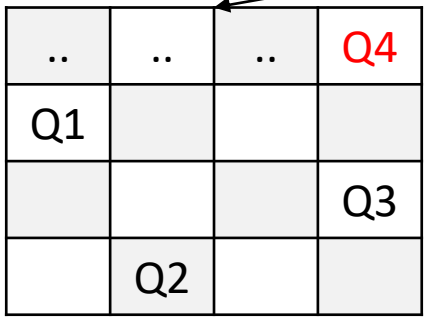


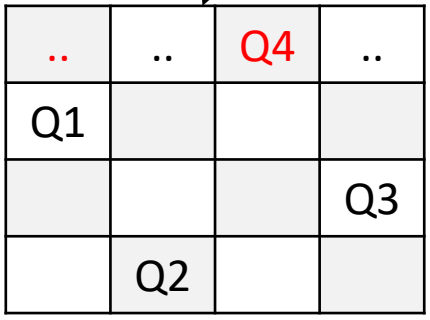






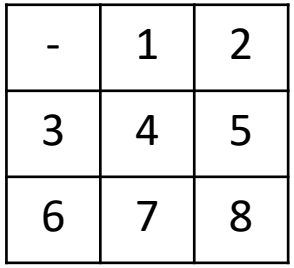




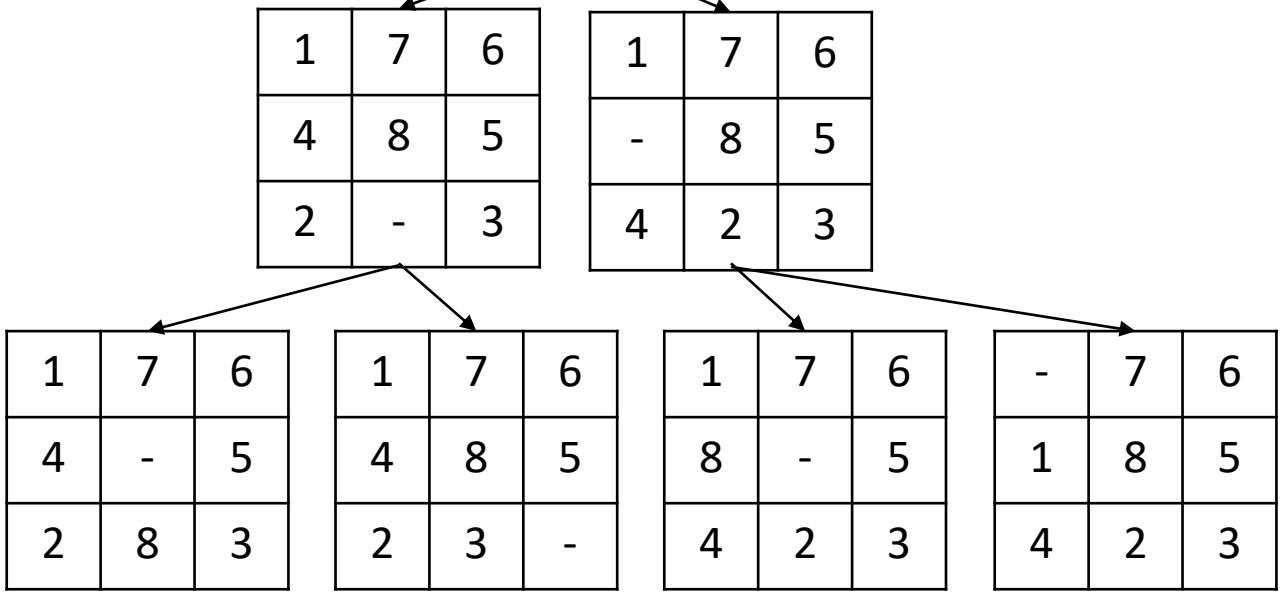


N-Tile

Initial State



Goal State



Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
<LOC, ID>	Move Empty to near by Tile		ID=LOC+1	Transition + Positional + Distance+ Other approaches	9!

PS

s_1 $h_1(n)$ & $h_2(n)$ Same info ✓ best

(s2) $h_3(n)$ & $h_4(n)$ → Combine $h_3(n) + h_4(n)$

$h_3(n)$ ↓ diff

$h_4(n)$ ↓ dim

Heuristic Design: Use case

Reference paper

- The paper uses A* and Greedy Best-First Search (GBFS) as the two main search algorithms to evaluate *learned heuristics* on several grid-based domains, specifically:
  - Sokoban
  - Maze with Teleports
  - Sliding Tile Puzzle
- These domains naturally represent states as grids, making them suitable for convolutional neural networks.
- The authors train neural networks to produce heuristic values and then use those heuristics inside A* and GBFS to solve the problems.

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Grid Domains Used in the Paper

- **Sokoban:**
  - A warehouse puzzle where an agent pushes boxes to target positions.
- **Maze with Teleports:**
  - A maze where the agent must reach a goal while optionally using teleport pairs.
- **Sliding Tile Puzzle:**
  - A classic N-puzzle where tiles are moved into a target arrangement.
  - All three can be encoded as 2D grids and processed by convolutional neural networks.

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Neural Network as the Heuristic Function

The paper represents the heuristic as:

$$h(s, \theta)$$

where:

- s = current grid state,
- θ = learned neural network parameters,
- $h(s, \theta)$ = predicted heuristic value.

For grid domains, the network contains:

- 7 convolution layers,
- 4 CoAt (Convolution + Attention) blocks,
- Mean pooling,
- A fully connected layer producing one scalar heuristic value.

How A* Uses the Learned Heuristic?

A* evaluates each node using:

$$f(n) = g(n) + h(n)$$

where:

- $g(n)$:actual path cost from the start,
- $h(n)$:*neural network estimate*,
- $f(n)$:total estimated solution cost.

At each step:

1. Generate successor states.
2. Compute $h(s, \theta)$ for each successor.
3. Compute $f(n)$.
4. Expand the node with the smallest $f(n)$.

How GBFS Uses the Learned Heuristic?

GBFS ignores the accumulated cost and chooses the state with the smallest heuristic:

$$f(n) = h(n)$$

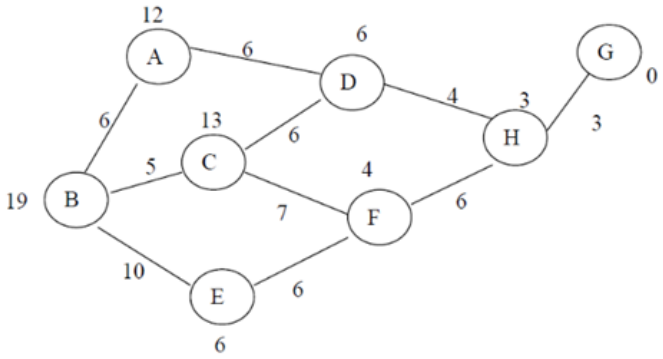
At each step:

- Generate successors.
- Predict heuristic values.
- Expand the state with the lowest predicted value.

✓ GBFS is often faster but does not guarantee optimal solutions.

Exercise 1

- Consider the following weighted graph. The number beside each node represents its heuristic value $h(n)$,which estimates the cost from that node to the goal node G.
- The start node is B, and the goal node is G.



- Apply GBFS on this graph.
- Apply A* Algorithm on this graph.
- Compare your findings from the above two
- Check if the heuristics is admissible and consistent.

Exercise 2

- A warehouse robot has to collect three packages and reach the exit.
- The warehouse has rooms connected by corridors. Some corridors have locked doors. The robot can move from one room to an adjacent room with cost 1. To pass through a locked door, the robot must first collect the matching key.
- **Warehouse layout** Rooms:

$$S, A, B, C, D, E, F, G, X$$

where:

- S= start room
- X= exit room
- Packages are located at C, F, and G
- Key $K_1$is located at B
- Key $K_2$is located at E

The robot must collect all three packages at C, F, and G, then reach X.

Exercise 2 Continued

Corridor	Cost	Condition
(S-A)	1	Open
(A-B)	1	Open
(A-C)	2	Open
(B-D)	2	Requires (K1)
(C-D)	2	Open
(D-E)	1	Open
(E-F)	2	Requires (K2)
(D-G)	3	Open
(G-X)	2	Open
(F-X)	2	Open

Part A: State Representation

Define a state representation for this problem.

Your state should include:

1.Current room of the robot.

2.Packages already collected.

3.Keys already collected.

Write the initial state and one possible goal state.

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Exercise 2 Continued

Part B: Relaxed Problems

A relaxed problem is obtained by removing one or more constraints from the original problem.

Consider the following relaxed versions of the warehouse problem:

- **Relaxation 1:** Ignore all locked-door constraints. The robot can pass through any corridor.
- **Relaxation 2:** Ignore package collection order. Estimate the cost only as the shortest distance from the current room to the exit.
- **Relaxation 3:** Assume the robot can collect any remaining package instantly once it enters the same connected component, but it must still move to the exit.
- **Relaxation 4:** Ignore the robot's current location and estimate only the minimum cost needed to connect all uncollected package locations and the exit.

Answer the following:

1. For each relaxed problem, explain which constraint has been removed.
2. Which relaxed problem is likely to give the weakest heuristic?
3. Which relaxed problem is likely to give the strongest heuristic?
4. Why does the solution cost of a relaxed problem give an admissible heuristic for the original problem?
5. Construct few heuristics by picking some relaxed version of the problem.

Thank You for your time & attention !

Slides are Licensed Under : CC BY-NC-SA 4.0